

AWS マイスターシリーズ

AWS SDK for Java/.NET Eclipse plugin, Visual Studio

2013.12.11

アマゾン データ サービス ジャパン株式会社

ソリューションアーキテクト

片山暁雄 渡邊源太

Agenda

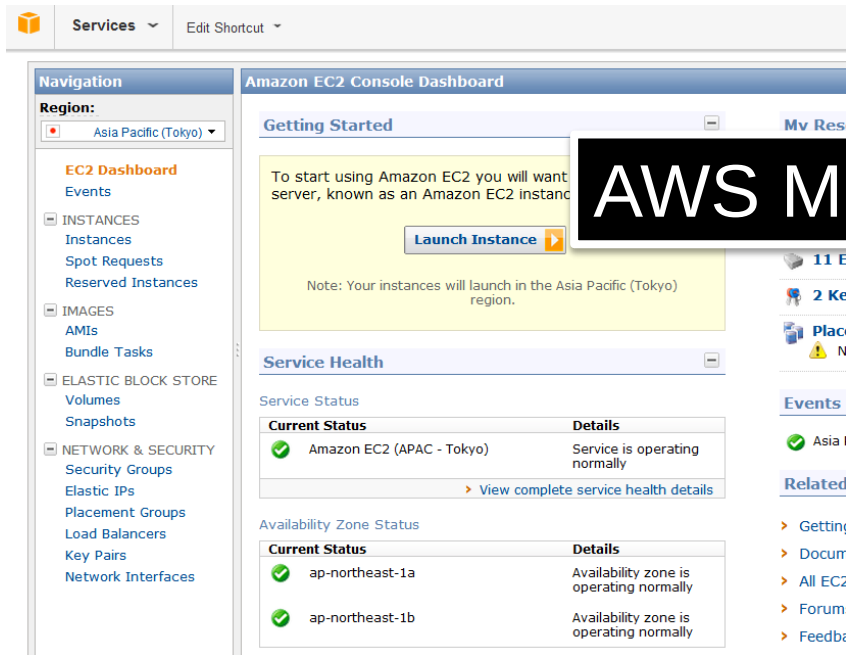
- 📦 AWS SDK オーバービュー
- 📦 AWS SDK for Java
- 📦 AWS SDK for .NET
- 📦 まとめ



AWS SDK オーバービュー



AWSのサービス操作とえば



AWS Management Console

AWSコマンドラインツール

```
awscli$ aws ec2 describe-instances --region ap-northeast-1 --query 'Reservations[*].Instances[*].InstanceId' --output text
i-10-150-18
t1.m
akiok 0
t1.m
ap-northeast-1a aki-44992845
10.150.186.176
avirtual xen 71cde051-e2c
dc1c83dd default
BLOCKDEVICE /dev/sda1 vol-
instance i-d5b4dad5
V1-y9qaQBDz8G
C:\¥>ec2-describe-instances
```

これらの裏側では・・・

- 📦 各サービスの各操作にAPIが定義されている
 - 📦 AWSでは・・・
 - ・ 人間がGUIまたはCUI越しに叩く
 - ・ プログラミングしてそれを自動化・簡易化・カスタマイズ
 - ・ 人間が手でやらなくてはいけない事をプログラミングして自動化できる、これがSDKで簡単に実現可能
- AWSの実は最も優れた一面

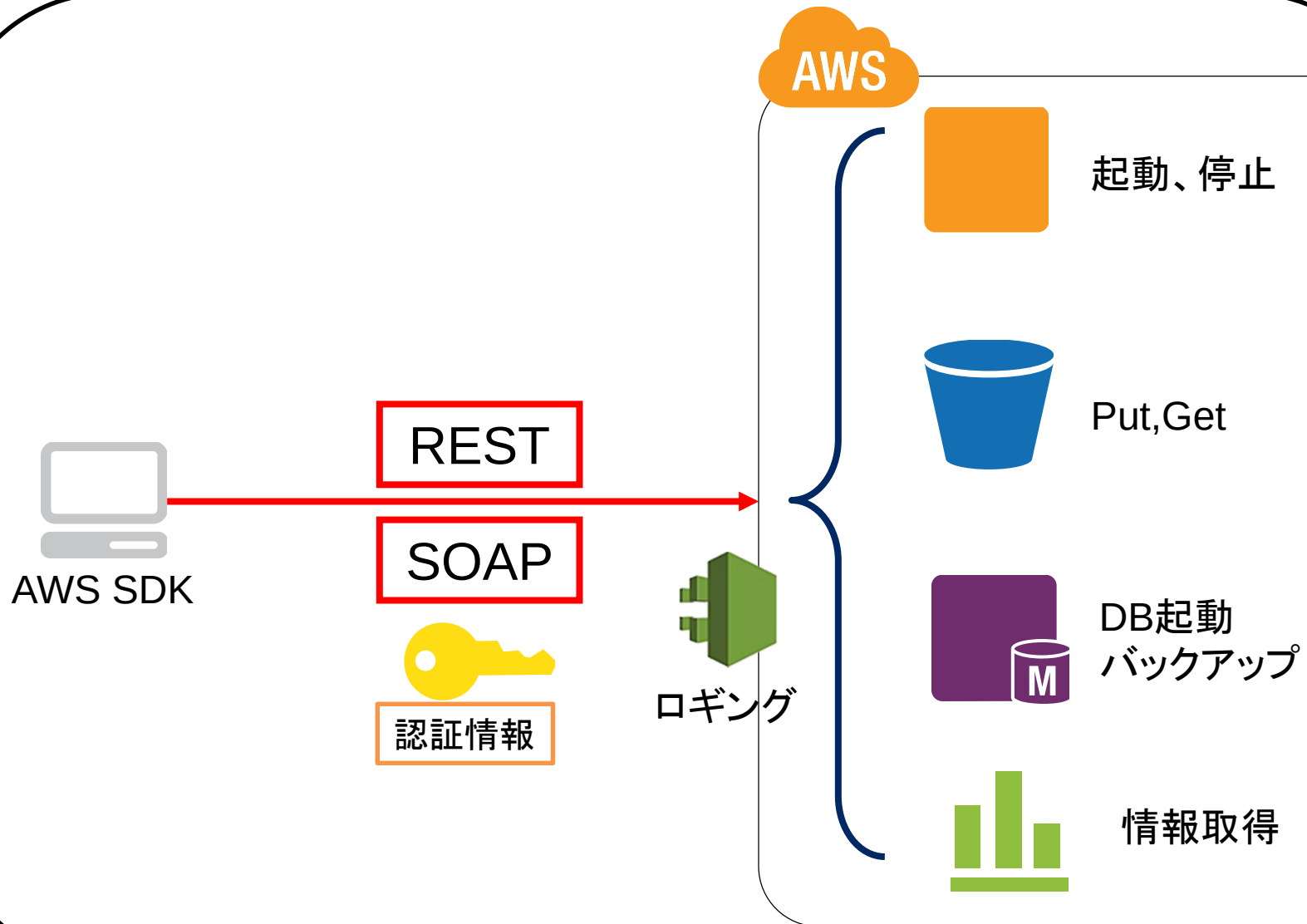
AWS SDKとは

AWSのサービスをプログラムで操作できるSDK

- さまざまな言語で
 - AWS SDK for Java
 - AWS SDK for .NET
 - AWS SDK for Ruby
 - AWS SDK for PHP
 - AWS SDK for Python (boto)
 - AWS SDK for node.js
 - AWS SDK for Android
 - AWS SDK for iOS
 - 有志の方による実装（ActionScript）も
- 通信は原則HTTPS



動作イメージ



認証情報

- 📦 AWSマネジメントコンソールのIAMタブで認証情報作成
- アクセスキー、シークレットキー、(optional:MFA)
 - シークレットキーは作成時のみ取得可能

The screenshot displays the AWS IAM console interface. At the top, there is a 'Viewing:' search bar. Below it is a table with columns 'User Name', 'Groups', and 'Password'. The table lists four users: 'AdminUser' (checked), 'adsj-contents', 'adsj-doc-share', and 'c9katayama'. Below the table, it indicates '1 Users Selected' and shows the selected user as 'User: AdminUser'. There are four tabs: 'Groups', 'Permissions', 'Security Credentials' (which is selected and highlighted with an orange bar), and 'Summary'. The 'Security Credentials' tab shows the 'Access Credentials' section, which is highlighted with a red rectangle. This section displays 'Access Keys' for the user, showing the key ID 'AKIAJNBYNVU2JAIPPYFQ', the status 'Active' in green, and the creation date '2012-02-18 08:06 UTC+0900'. A 'Manage Access Keys' button is located at the bottom of this section.

User Name	Groups	Password
☒ AdminUser	0	✓
☐ adsj-contents	0	
☐ adsj-doc-share	0	
☐ c9katayama	0	

1 Users Selected

User: AdminUser

Groups Permissions **Security Credentials** Summary

Access Credentials

Access Keys: AKIAJNBYNVU2JAIPPYFQ
Active
2012-02-18 08:06 UTC+0900

Manage Access Keys

操作の種類

例：EC2

- インスタンス起動・・・RunInstances
- リブート・・・RebootInstances
- IPアドレス付与・・・AllocateAddress
- などさまざまな操作が、プログラムから実行可能
- <http://docs.amazonwebservices.com/AWSEC2/latest/APIReference/Welcome.html>
- SDKによってサポートする操作が異なる

 APIコールに対する制限は、Identity and Access Management(IAM) のドキュメントを参照

AWS SDK for Java



AWS SDK for Java



- 📦 AWS提供の開発用Java SDK
 - <http://aws.amazon.com/sdkforjava/>
 - <https://github.com/aws/aws-sdk-java>
- 📦 環境: Java6以降
- 📦 最新版:1.6.8
- 📦 依存ライブラリ
 - stax
 - aspectj
 - commons-codec
 - commons-logging
 - freemarker
 - httpcomponents-client
 - jackson
 - java-mail
 - spring



サポートしているサービス

Compute & Networking

-  [AWS Direct Connect »](#)
-  [Amazon EC2 »](#)
-  [Elastic Load Balancing »](#)
-  [Auto Scaling »](#)
-  [Amazon EMR »](#)
-  [Amazon Route 53 »](#)
-  [Amazon VPC »](#)

Database

-  [Amazon DynamoDB »](#)
-  [Amazon RDS »](#)
-  [Amazon Redshift »](#)
-  [Amazon ElastiCache »](#)
-  [Amazon SimpleDB »](#)

Storage & Content Delivery

-  [Amazon S3 »](#)
-  [Amazon Glacier »](#)
-  [Amazon CloudFront »](#)
-  [AWS Storage Gateway »](#)
- [AWS Import/Export »](#)

Deployment & Management

-  [AWS Elastic Beanstalk »](#)
-  [AWS CloudFormation »](#)
-  [Amazon CloudWatch »](#)
-  [AWS Data Pipeline »](#)
-  [AWS Identity and Access Management »](#)
-  [AWS OpsWorks »](#)

App Services

-  [Amazon Elastic Transcoder »](#)
-  [Amazon SQS »](#)
-  [Amazon SNS »](#)
-  [Amazon SES »](#)
-  [Amazon SWF »](#)
-  [Amazon CloudSearch »](#)

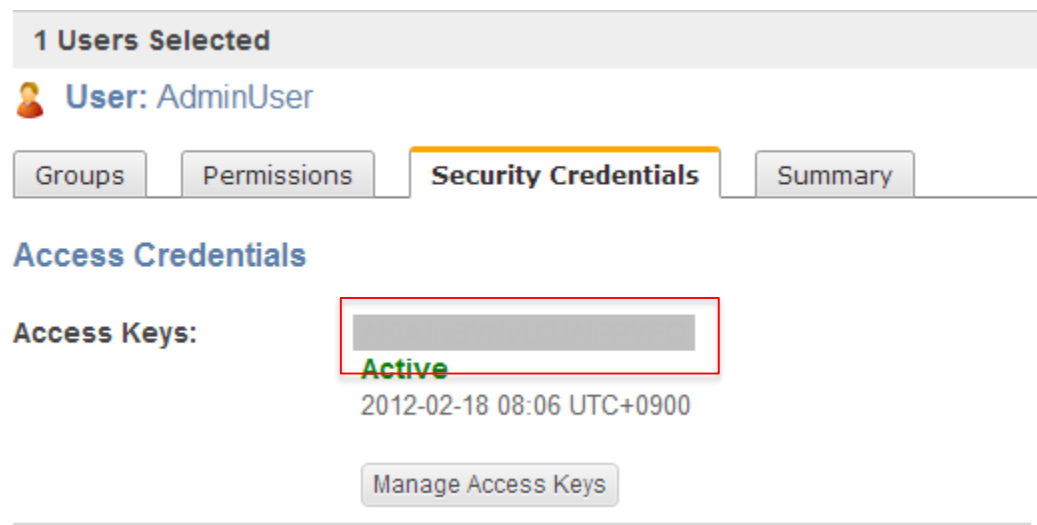
使い方

SDKをダウンロード

- クラスパスにjarファイルを設定

認証情報の取得

- AWSマネジメントコンソールからIAMユーザーを作成し、クレデンシャル情報（Access Key, Secret Access Key）を取得



AWS Toolkit for Eclipse

- ❏ EclipseにAWS SDKを使ったプロジェクトを追加するプラグイン
- ❏ AWSを使用したアプリの開発/テストを効率化
- ❏ EC2やS3などのサービス管理コンソールも付属



Java SDKクライアントクラスの基本

クレデンシャル情報を、クライアントクラスに設定

- アクセスキー
- シークレットアクセスキー

設定方法：

- BasicAWSCredentialsで直接キーをコードに入れる（非推奨）
- PropertiesCredentialsでプロパティファイルに記述する
- STS(一時トークン) を取得して設定
- IAM Roleを利用する

コード例

```
AWSCredentials credentials = new null;  
try {  
    credentials = new PropertiesCredentials(Util.class  
        .getResourceAsStream("AwsCredentials.properties"));  
} catch (IOException e) {  
    //例外処理  
}  
//このクレデンシャルを使ってClientを生成する
```

IAM利用時は、IAMユーザーのアクセスキー、シークレットキーを利用。
IAMユーザーでアクセス権のないメソッドを呼ぶと、例外が発生。



リージョンの設定

📦 デフォルトはus-east

📦 クライアントクラスへの設定方法は2種類

- setEndpointメソッドで指定（旧来の方法）
 - setEndpoint("ec2.ap-southeast-1.amazonaws.com")
 - サービス毎のエンドポイントは下記URL参照
 - <http://docs.amazonwebservices.com/general/latest/gr/rande.html>
- setRegionメソッドで指定（新しい指定）
 - setRegion(Region.getRegion(Regions.AP_NORTHEAST_1));



SetとWith

📦 API呼び出しパラメータのデータセットは、set～とwith～メソッドがある

- 例： setInstanceType() と withInstanceType()
- set ～は、通常のJavaのsetterとして利用
- with ～は、戻り値としてパラメータのインスタンスを返す
 - with ～. with ～というように、メソッドを連続で呼び出せる

```
// 立ち上げたいインスタンス情報の作成
RunInstancesRequest req =
    new RunInstancesRequest("ami-xxxxx", 5, 5);

req.setKeyName("yourkeyname")
req.withSecurityGroupIds("yoursecgroup")
    .withMonitoring(true)
    .withInstanceType(InstanceType.M1Small);

// インスタンスの起動
ec2.runInstances(runInstancesRequest);
```

EC2



AmazonEC2Client

```
// EC2操作のクライアント
AmazonEC2 ec2
    = new AmazonEC2Client(credentials);
// リージョン設定
ec2.setRegion(Region.getRegion(Regions.AP_NORTHEAST_1));

// 立ち上げたいインスタンス情報の作成
RunInstancesRequest runInstancesRequest = new
RunInstancesRequest("ami-xxxxx", 5, 5)
    .withKeyName("yourkeyname")
    .withSecurityGroupIds("yoursecgroup")
    .withMonitoring(true)
    .withInstanceType(InstanceType.M1Small);

// インスタンスの起動
ec2.runInstances(runInstancesRequest);
```

S3



AmazonS3Client

```
// S3操作用クライアント  
AmazonS3 s3 = new AmazonS3Client(credentials);  
  
// bucket作成  
s3.createBucket("mybucket");  
  
//オブジェクトのPUT  
PutObjectResult ret = client.putObject("mybucket",  
"aaa.txt", file);
```

S3での巨大ファイルの分割アップロード

```
//TransferManagerを作成
```

```
AmazonS3Client client = new AmazonS3Client(cred);
```

```
TransferManager manager = new TransferManager(client);
```

```
//最低1チャンクを5MBに設定
```

```
TransferManagerConfiguration c =
```

```
    new TransferManagerConfiguration();
```

```
c.setMinimumUploadPartSize(5 * 1024L * 1024L);
```

```
manager.setConfiguration(configuration);
```

```
//巨大ファイルのアップロード実行と進捗の監視
```

```
Upload upload =
```

```
    manager.upload(bucketName, bigfileName, target, o);
```

```
while (upload.isDone() == false) {...
```



SDKならではの機能

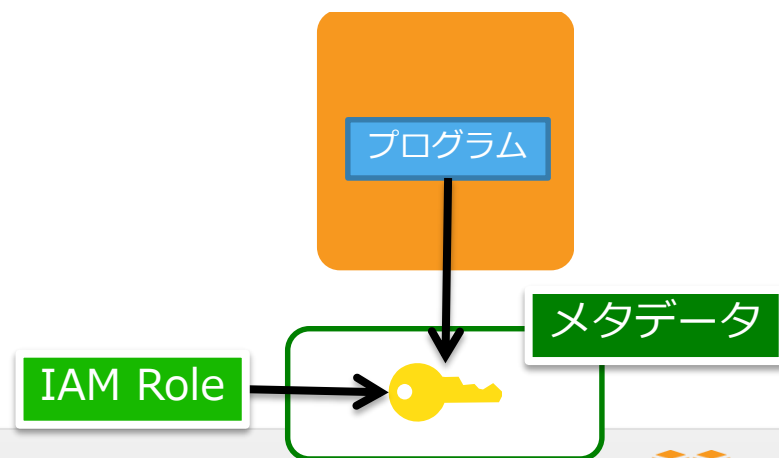
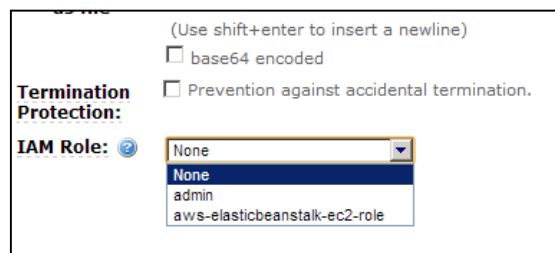
- 📦 S3のクライアントサイド暗号化
- 📦 DynamoDBのオブジェクトマッパー
 - Plain Old Java Object (POJOs)とDynamoDBのテーブルをマッピング
- 📦 Amazon SQS クライアントサイドバッファリング
 - メッセージをクライアント側でバッファリング。パフォーマンスの改善
- 📦 AWS SimpleWorkflowService FlowFramework
- 📦 IAMロールを使った認証



IAM Role for EC2 instances

EC2インスタンスに、指定のロールを付与する機能

- EC2起動時にロールを指定すると、認証情報がメタデータに設定される
- 認証情報はSTS(Security Token Service)で生成
 - インスタンス毎に異なるキー
 - 有効期限付きで、期限が来るとローテート
- アプリケーションから認証情報を取得し、AWSサービスへアクセス
 - インスタンス内からメタデータにアクセス
 - アクセスキーID、シークレットアクセスキー、セッショントークンを取得
 - 3つの認証情報でAPI呼び出し



IAM Roleを使った認証



- ❏ EC2起動時に、IAM Roleを設定
- ❏ インスタンス上でSDKを利用。認証設定なしでAPIコール可能
 - ・ インスタンスメタデータから、STS(AWS Security Token Service) をSDKが自動取得

```
AWSCredentials credentials =  
    new BasicAWSCredentials("アクセスキー","シークレットキーID");  
AmazonEC2 ec2 = new AmazonEC2Client(credentials);  
ec2.describeInstances();
```



IAM Role利用後

```
AmazonEC2 ec2 = new AmazonEC2Client();  
ec2.describeInstances();
```

IAM Role適用のインスタンス上では、認証情報の設定が不要

MFAデバイスを使ったAPIコール

- 📦 APIコール時に、MFAデバイスでの認証を必須とする機能
- 📦 以下の流れで利用
 1. IAMユーザーを作成
 2. IAMユーザーのMFAデバイスを登録
 3. IAMユーザーに対して、ポリシーを登録
 - API呼び出しにMFAが必要とする制限を付与
 4. AWSSecurityTokenServiceClientを使用して、一時トークンを取得
 - ここでMFAデバイスのコードを使用
 5. 以降のAPIコールに、取得した一時トークンを使用



IAMポリシー設定例

```
{ "Version": "2012-10-17",  
  "Statement": [  
    { "Action": ["ec2:*"],  
      "Effect": "Allow",  
      "Resource": ["*"],  
      "Condition": {  
        "NumericLessThan": { "aws:MultiFactorAuthAge": "3600" }  
      }  
    ]  
  }  
}
```

MFA認証後1時間(3600秒)以内のAPIコールのみ許可



コード例

```
// 認証情報の読み込み
AWSCredentials credentials = new BasicAWSCredentials("アクセスキー","シークレットキー");
// Security Token Service呼び出し
AWSSecurityTokenServiceClient stsClient =
    new AWSSecurityTokenServiceClient(credentials);
GetSessionTokenRequest stsRequest = new GetSessionTokenRequest();
stsRequest.setDurationSeconds(3600);
stsRequest.setSerialNumber("GAKTXXXXXXXXX");//登録したMFAデバイスのシリアルコード
stsRequest.setTokenCode("000000");//MFAデバイスの数値

GetSessionTokenResult stsResult = stsClient.getSessionToken(stsRequest);
Credentials stsCredentials = stsResult.getCredentials();

BasicSessionCredentials credentials = new BasicSessionCredentials(
    stsCredentials.getAccessKeyId(),
    stsCredentials.getSecretAccessKey(),
    stsCredentials.getSessionToken());

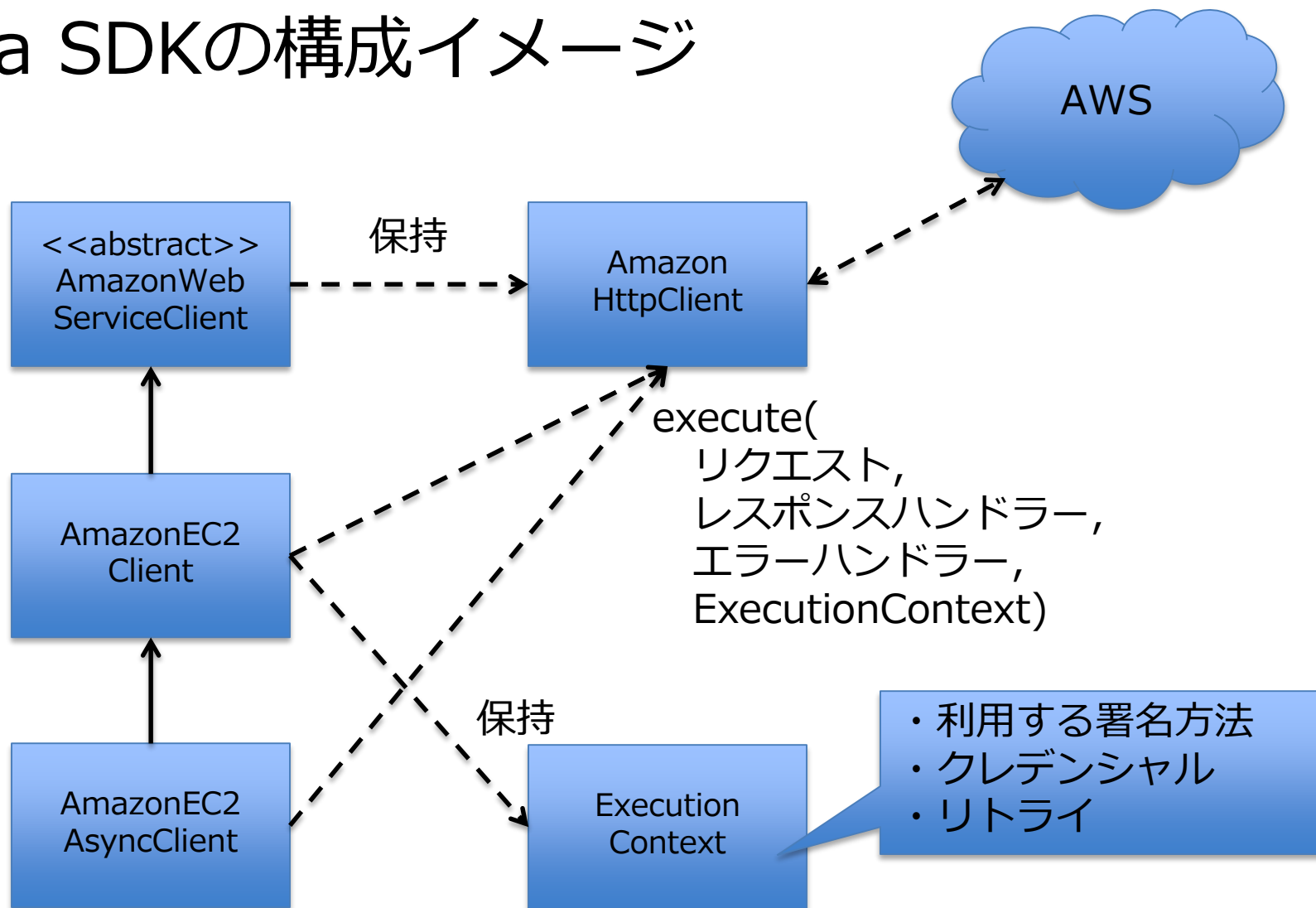
// EC2クライアントの作成
AmazonEC2Client ec2 = new AmazonEC2Client(credentials);
//以下通常のAPIコール
```



Java SDKの構成

- 📦 各サービス毎にクライアントクラスが付属
 - 面倒な下記のようなことは利用者から隠ぺいしてくれる
 - HTTP通信及び例外ハンドリング
 - 署名
 - エラー時のリトライ処理
- 📦 サービスどれでもほぼ同一で直感的な使い勝手の提供
- 📦 極力、利用者は単純に呼び出すだけでよい
- 📦 プラガブルな構成で、部分的に入れ替えられる
 - 挙動を変えたい
 - 例外のハンドリング方法を変えたい
 - 処理速度を上げたい

Java SDKの構成イメージ



Java SDKのパッケージ構成

com.amazonaws

- auth : 署名関連
- handlers : SDKの挙動をカスタマイズするハンドラー
- http : 実際のHTTPリクエストを処理する
- internal : リトライ関連
- sdk : バージョン情報
- services :
 - 各サービス毎の実体（クライアントクラス、データクラス）
- transform : API経由で返ってくるXMLをパースする実体
- util : ユーティリティ

各サービスのクライアントクラスは実体が2つ

- 同期通信を中心としたクライアント : XxxClient
- 非同期通信を追加した非同期クライアント : XxxAsyncClient

SDKの有効な利用方法

利用方法（運用面）

- 決まったインスタンスを起動するバッチ
- 固定ディスク（EBS）のスナップショットを定期取得
- S3からデータを定期的に取得
- S3上のデータを監視して、無くなったら通知

利用方法（アプリから）

- アプリのバックエンドとして、DynamoDBを使う
- メール送信のためにSMSを使う
- データ保存のためにS3を使う

アドバンスドなトピック

- 📦 Java SDKは拡張性をかなり確保した状態になっている
- 📦 Client/AsyncClient自体も自分で拡張できる
 - インターフェースをそのまま実装する
- 📦 細かく拡張ポイントを入れ替えることも可能
 - ClientConfiguration
 - RequestHandler
 - CustomBackoffStrategy
 - Marshaller/Unmarshaller

ClientConfiguration

各Clientクラスの基本的な挙動を決定する設定

- ソケットのタイムアウト値
- 最大コネクション数
- 最大リトライ数
- 使用するプロトコル(デフォルトHTTPS)
- プロキシ関連設定
- コネクションタイムアウト

各Client生成時にClientConfigurationを渡す

RequestHandler

- 📦 ServletでいうところのいわゆるFilter
 - デバッグ時や、何か統一的に処理させたい場合など便利
 - beforeRequest : リクエスト送信前
 - afterResponse : レスポンス受信後
 - afterError : エラー受信後
- 📦 各サービスごとに定義可能
 - Clientで普通に渡す
 - com/amazonaws/services/xxx/request.handlersというファイルにおいて、これに定義するとロードされる
- 📦 標準で既に定義されているものも
 - EC2RequestHandler
 - QueueUrlHandler
 - Route53IdRequestHandler

CustomBackoffStrategy

- 📦 AWSクラウド側へのリトライ間隔をコントロールする
 - カスタマイズして現状使っているのはDynamoDBのみ
 - 実行コンテキストであるExecutionContextに設定して渡す
- 📦 AmazonHttpClientの600行目付近
 - `pauseExponentially`メソッド

```
private void pauseExponentially(int retries, AmazonServiceException
previousException, CustomBackoffStrategy backoffStrategy) {
    long delay = 0;
    if (backoffStrategy != null) {
        delay = backoffStrategy.getBackoffPeriod(retries);
    } else {
        long scaleFactor = 300;
        ...
        delay = (long) (Math.pow(2, retries) * scaleFactor);
    }
    ...
}
```

Marshaller/Unmarshaller

- 📦 AWSが提供しているXMLパース(一部JSONパース)では遅いので色々いらないものを省略してパースしたい
 - Java SDKではStAXというプルモデル型のパーサー
 - Clientを書いて、自分でMarshaller/Unmarshallerする
- 📦 各サービス.model.transformの下
 - ここにMarshaller/Unmarshallerが大量にいる
- 📦 コーディングポリシー的には
 - Clientの各メソッド内でMarshallしてRequest<X>を作成
 - UnmarshallerはResponseHandlerにセットされて、AmazonHttpClientに渡されてレスポンス帰ってきたら実行

AWS SDK for .NET



AWS SDK for .NET



AWS SDK for .NET

- Amazon提供のAWS開発用.NET SDK
- <http://aws.amazon.com/sdkfornet/>
- 環境 :
 - .NET Framework 3.5以降
 - Visual Studio 2010以降
- 最新版2.0.2
- C#およびVisual Basicをサポート



AWS SDK for .NET V2

- ❏ Windows StoreおよびWindows Phone 8アプリおよび非同期処理をサポートしたあたらしいバージョンのSDK
- ❏ .NET 4.5をターゲットにしたタスクベースの非同期パターンのサポート
- ❏ ターゲットとするプラットフォームごとに別個のアセンブリを提供
- ❏ あたらしいS3暗号化クライアントの提供



サポートしているサービス

Compute & Networking

-  AWS Direct Connect »
-  Amazon EC2 »
-  Elastic Load Balancing »
-  Auto Scaling »
-  Amazon EMR »
-  Amazon Route 53 »
-  Amazon VPC »

Database

-  Amazon DynamoDB »
-  Amazon RDS »
-  Amazon Redshift »
-  Amazon ElastiCache »
-  Amazon SimpleDB »

Storage & Content Delivery

-  Amazon S3 »
-  Amazon Glacier »
-  Amazon CloudFront »
-  AWS Storage Gateway »
-  AWS Import/Export »

Deployment & Management

-  AWS Elastic Beanstalk »
-  AWS CloudFormation »
-  Amazon CloudWatch »
-  AWS Data Pipeline »
-  AWS Identity and Access Management »
-  AWS OpsWorks »

App Services

-  Amazon Elastic Transcoder »
-  Amazon SQS »
-  Amazon SNS »
-  Amazon SES »
-  Amazon SWF »
-  Amazon CloudSearch »

AWS SDK for .NETに含まれるもの

- 📦 AWS Toolkit for Microsoft Visual Studio
- 📦 Visual Studioプロジェクトテンプレート
- 📦 AWS Tools for Windows PowerShell
- 📦 AWS .NETライブラリ
- 📦 C#コードサンプル
- 📦 ドキュメント



インストール方法

- ❏ 以下のページの右上隅にある「AWS .NET for SDK」ボタンをクリック
 - <http://aws.amazon.com/jp/sdkfornet/>
- ❏ ファイルを保存するかどうかをたずねるメッセージがブラウザに表示されたら、ローカルのディスクに保存
- ❏ 保存したインストーラを開いてインストールプロセスを開始

ダウンロード

AWS SDK for .NET »

[GitHub でソースを取得する »](#)

AWS Toolkit for
Microsoft Visual Studio »



AWS Toolkit for Visual Studio



- 📦 Microsoft Visual Studioを使用するサービスの管理が可能
- 📦 AWS SDK for .NETによるアプリケーション開発に対応
- 📦 AWS Elastic Beanstalk/AWS CloudFormationによる.NETアプリケーションのデプロイに対応

機能

- 📦 アカウント管理
- 📦 AWS Explorer
 - 各種サービスの情報表示/操作
- 📦 EC2 コンソール
 - EC2のコントロールパネル
- 📦 AWS CloudFormation
- 📦 AWS Elastic Beanstalk
 - デプロイ、デバッグ、環境構築

EC2 コンソール

AMI管理

- 検索、削除、AMIからの起動

インスタンス管理

- 右クリックから状態変更、AMI作成、EBSの操作
- インスタンスタイプおよびセキュリティグループによるフィルター

The screenshot displays the AWS Management Console interface for the Asia Pacific (Tokyo) region. The top navigation bar shows 'Asia Pacific (Tokyo) EC2 Instances' and 'Asia Pacific (Tokyo) EC2 AMIs'. The left sidebar contains a list of AMIs under the heading 'Asia Pacific (Tokyo) EC2 AMIs'. The main content area shows a table of EC2 instances. A context menu is open over the instance 'Active Directory サーバー #2', displaying various actions that can be performed on the instance.

AMI ID	AMI Name	Description
ami-0e5eed0f	Lab CRM Image	Lab CRM AMI
ami-16b60717	Clone Windows Instance	
ami-52239253	VM Import	VM Import Instance
ami-a0ab19a1	Test	

Name	Instance ID	Status	AMI ID	Root Device Type	Type	Security Groups	Zone	Launch Time
Active Directory サーバー #1	i-cfcc11cf	stopped	ami-ecea5ced	ebs	m1.small	Back-End-Windows	ap-northeast-1a	24/07/11
Active Directory サーバー #2	i-69e66d69	stopped	ami-80a01181	ebs	m1.small	Back-End-Windows	ap-northeast-1b	24/07/11
SharePointサーバー #1	i-eb792feb	stopped	ami-80a01181	ebs	m1.small	Back-End-Windows	ap-northeast-1a	24/07/11
SharePointサーバー #2	i-fb6c3afb	stopped	ami-80a01181	ebs	m1.small	Back-End-Windows	ap-northeast-1b	24/07/11
SQL Server プリンシパル	i-f56d3bf5	stopped	ami-80a01181	ebs	m1.small	Back-End-Windows	ap-northeast-1a	24/07/11
SQL Server ミラー	i-1798ca17	stopped	ami-80a01181	ebs	m1.small	Back-End-Windows	ap-northeast-1b	24/07/11
SQL Server Witness	i-3f44193f	stopped	ami-80a01181	ebs	m1.small	Back-End-Windows	ap-northeast-1a	24/07/11
	i-71476471	running	ami-80a01181	ebs	m1.small	Back-End-Windows	ap-northeast-1b	24/07/11

Context Menu Options:

- Get System Log
- Create Image (EBS AMI)
- Change Termination Protection
- View/Change User Data
- Change Instance Type
- Change Shutdown Behavior
- Terminate
- Reboot
- Stop
- Start
- Properties

AWS CloudFormation

- ❏ AWS CloudFormationテンプレートによるアプリケーションのデプロイ
- ❏ Single Instance Template またはLoad Balanced Templateから選択
- ❏ .NET Frameworkのバージョンが指定可能
- ❏ Application URLからアプリにアクセス可能

Publish to AWS

AWS Options
Set Amazon EC2 and AWS CloudFormation options for the deployed application.

Amazon EC2

Use custom AMI: Instance type *: Micro
Key pair *: gentaw Security group *: Windows

AWS CloudFormation

Stack name: PetboardVS2010
✓ Stack name is valid and available
SNS Topic: Create New Topic
Creation timeout: None ☒ Rollback on failure

Cancel Back Next Finish

Refresh Connect to Instance

Stack Name: PetboardVS2010 Created: 10/29/2012 3:44:12 PM
Status: **CREATE_COMPLETE** Create Timeout: None
Status (Reason): ☒ Rollback on Failure
SNS Topic:
Description: This will create a Load Balancer and Auto Scaling Group and deploy your application to it. **WARNING** This template creates one or more Amazon EC2 instances. You will be billed for the AWS resources used if you create a stack from this template.
Application URL: <http://PetboardV-ElasticL-115W944FJG9JL-1818755262.ap-northeast-1.elb.amazonaws.com>

Events

EC2 Instances:

Instance	Status	AMI ID	Type	Tags
i-e608abe5	running	ami-5c44b5d1	micro	awscloudformation:stack-name=PetboardVS2010, aws:autoscaling:groupName=PetboardVS2010

Template Resources:

Logical ID	Physical ID	Type	Status
DeployedApplicationWaitHandle	https://cloudformation-waitcondition	AWS::CloudFormation::WaitConditionHandle	CREATE_COMPLETE
PetboardVS2010-LaunchConfig-1	AWS::AutoScaling::LaunchConfiguration	CREATE_COMPLETE	
ElasticLoadBalancer	PetboardV-ElasticL-115W944FJG9JL	AWS::ElasticLoadBalancing::LoadBalancer	CREATE_COMPLETE

Outputs:

Name	Value	Availability Zones
Instances	PetboardV-ElasticL-115W944FJG9JL	ap-northeast-1a, ap-northeast-1b, a
DNS Name	PetboardV-ElasticL-115W944FJG9JL-1818755262.ap-northeast-1.elb.amazonaws.com	Canonical Hosted Zone ID Z2YN177SR11GT

.NET SDKにおけるアクセスキー管理

📦 Service Clientにクレデンシャルを設定

```
var accessKey = ""; // Get access key from a secure store  
var secretKey = ""; // Get secret key from a secure store  
var s3Client =  
    AWSClientFactory.CreateAmazonS3Client(accessKey, secretKey, R  
        egionEndpoint.APNortheast1;
```

📦 アプリケーションコンフィグ（App.config）に記述

📦 IAM Roleを利用する



App.configの設定

- App.configにアクセスキーとシークレットキーを以下のように設定

```
<configuration>  
  <appSettings>  
    <add key="AWSAccessKey" value="[ENTER YOUR ACCESS  
KEY ID HERE]"/>  
    <add key="AWSSecretKey" value="[ENTER YOUR SECRET  
ACCESS KEY HERE]"/>  
  </appSettings>  
</configuration>
```


リージョンの設定

- ❏ リージョンの選択は.NET SDK V2では必須
- ❏ App.configで指定

```
<configuration>  
  <appsettings>  
    <add key="AWSRegion" value="ap-northeast-1">  
  </add> </appsettings>  
</configuration>
```

- ❏ RegionEndPointパラメータで指定
 - var s3Client = new
AmazonS3Client(RegionEndpoint.APNortheast1);

- ❏ サービス毎のエンドポイントは下記URL参照
 - <http://docs.amazonwebservices.com/general/latest/gr/rande.html>



.NET SDK V2でサポートされない機能

- 📦 V1でサポートされていたWithメソッドは使用できない

```
TransferUtilityUploadRequest uploadRequest = new  
TransferUtilityUploadRequest()  
    .WithBucketName("my-bucket")  
    .WithKey("test")  
    .WithFilePath("c:¥test.txt");
```

- 📦 Withメソッドのかわりにコンストラクタ初期化子（Constructor Initializer）をつかう

```
TransferUtilityUploadRequest uploadRequest = new  
TransferUtilityUploadRequest  
{  
    BucketName = "my-bucket",  
    Key = "test",  
    FilePath = "c:¥test.txt"  
};
```



サンプル：EC2インスタンス起動



```
// EC2操作のクライアント
var ec2Client = new AmazonEC2Client();

// 立ち上げたいインスタンス情報の作成
var runInstanceRequest = new RunInstancesRequest{
    ImageId = "ami-xxx",
    MaxCount = 1,
    MinCount = 1,
    KeyName = "YourKeyName",
    SecurityGroups = { "yourSecurityGroup" },
    InstanceType = InstanceType.M1Small
};

// インスタンスの起動
ec2Client.RunInstances(runInstanceRequest);
```

サンプル : S3バケットの作成



```
// S3操作クライアント
var s3Client = new AmazonS3Client();

// bucket情報の設定
var putBucketRequest = new PutBucketRequest
{
    BucketName = "Mybucket",
    BucketRegion = S3Region.APN1
};

// bucket作成
s3Client.PutBucket(putBucketRequest);
```

サンプル : S3オブジェクトのアップロード

```
// S3操作クライアント
var s3Client = new AmazonS3Client();

//オブジェクト情報の設定
var putObjectRequest = new PutObjectRequest
{
    BucketName = "Mybucket",
    Key = "Item1",
    FilePath = "contents.txt"
};

//
var response = s3Client.PutObject(putObjectRequest);
```



サンプル：S3オブジェクトのアップロード (非同期)

```
// 非同期アップロード用のメソッド
async Task UploadFile(string bucketName, string filepath)
{
    var s3Client = new AmazonS3Client();
    var request = new PutObjectRequest()
    {
        BucketName = bucketName,
        FilePath = filepath
    };

    await s3Client.PutObjectAsync(request);
    Console.WriteLine("File Uploaded");
}
```

```
// UploadFileメソッドの呼び出し
await UploadFile(bucketName, filepath);
```



サンプル : RDSインスタンスの作成

```
// RDS操作用クライアント
```

```
var rdsclient = new AmazonRDSClient();
```

```
// DB instance情報の設定
```

```
var createDBInstanceRequest = new CreateDBInstanceRequest  
{  
    AllocatedStorage = 30,  
    DBInstanceClass = "db.m1.small",  
    DBInstanceIdentifier = "sqlserver",  
    Engine = "sqlserver-ex",  
    MasterUsername = "sa",  
    MasterUserPassword = "password"  
};
```

```
// DB instance作成
```

```
rdsclient.CreateDBInstance(createDBInstanceRequest);
```



.NET SDKの特徴

- Amazon DynamoDB オブジェクト永続フレームワーク
 - .NETクラスをAmazon DynamoDBの項目にマッピングしてデータを格納および取得
- Amazon S3 TransferUtility
 - マルチスレッド化されたAmazon S3マルチパートアップロード
- Amazon Glacier Archive TransferManager
 - 高レベルAPIによる自動的なファイル分割
- Amazon DynamoDB Session State Provider
 - ASP .NETのセッション状態をDynamoDBに格納



サンプル: Amazon DynamoDBオブジェクト永続フレームワークの利用(1/2)

- 📦 DynamoDBのテーブルに"Book"タイプを設定
- 📦 ハッシュキーは"Id", "Title", "Authors", "Price"

```
[DynamoDBTable("Books")]  
class Book  
{  
    [DynamoDBHashKey]  
    public int Id { get; set; }  
    public string Title { get; set; }  
    public List Authors { get; set; }  
    public double Price { get; set; }  
}
```

サンプル: Amazon DynamoDBオブジェクト永続化フレームワークの利用(2/2)

DynamoDBContextから”Book”タイプを利用

```
var client = new AmazonDynamoDBClient();
DynamoDBContext context = new DynamoDBContext(client);

// アイテムの格納
Book book = new Book
{
    Title = "Cryptonomicon",
    Id = 42,
    Authors = new List { "Neal Stephenson" },
    Price = 12.95
};
context.Save(book);

// アイテムの取得
book = context.Load(42);
Console.WriteLine("Id = {0}", book.Id);
Console.WriteLine("Title = {0}", book.Title);
Console.WriteLine("Authors = {0}", string.Join(", ", book.Authors));
```

AWS SDK for .NET

プラットフォームによる差異

- 📦 AWS SDK for .NET Framework 3.5
 - V1の機能および同様の非同期処理をサポート
- 📦 AWS SDK for .NET Framework 4.5
 - V1の機能に加えてC# 5.0のasync/awaitによるタスクベースの非同期処理
- 📦 AWS SDK for Windows RT
 - 非同期処理はasync/awaitのみ
 - Amazon S3とAmazon DynamoDBの一部機能が利用不可
 - Transfer Utility
 - IO Namespace
 - GetDecryptedPasswordによるWindowsパスワード取得が利用不可
- 📦 AWS SDK for Windows Phone 8
 - Windows RTと同様の制限
 - AWS SDK for Android/iOSと同等のサービスをサポート



まとめ

SDKを使うことで

- 運用管理での煩雑な手間をコーディングで楽にすることが可能
- どのサービスでも同じような使い勝手で利用可能
- 実際のサービス開発のお供に
 - S3
 - SNS/SQS/SWF
 - DynamoDB

AWSはコーディングする方の力を最大限に引き出すインフラ

- **HAPPY CODING!!**

参考情報

- 📦 AWS SDK for .NET
 - <http://aws.amazon.com/jp/sdkfornet/>
- 📦 AWS SDK for .NET API Reference
 - <http://docs.aws.amazon.com/sdkfornet/latest/apidocs/Index.html>
- 📦 AWS Forum development in .NET
 - <https://forums.aws.amazon.com/forum.jspa?forumID=61>
- 📦 AWS blog .NET Development
 - <http://blogs.aws.amazon.com/net/>